CS388G Problem Set #4

Nidhi Kadkol (nk9368)

April 2019

1(a)

Proof. We first show that for a subset U, $U \oplus X$ is good is equivalent to U being feasible.

$U \oplus X$ is a good subset $\iff$ $(U \oplus X) \oplus X$ is feasible $\iff$ $U \oplus (X \oplus X)$ is feasible (since $\oplus$ is associative) $\iff$ $U \oplus \phi$ is feasible $\iff$ U is feasible.

Since U is a subset of V and X and Y are a bipartition of V, each element of U comes from either X or Y.

Let $U = X' \cup Y'$ where $X' \subseteq X$, $Y' \subseteq Y$.

$w(U \oplus X) = w(X - U) + w(U - X) = w(X) - w(X') + w(Y')$

$v(U) = w(X \cap U) - w(Y \cap U) = w(X') - w(Y')$

Thus, $w(U \oplus X) = w(X) - v(U)$

Since $w(X)$ is constant, a maximum value of $w(X)$ $\iff$ a minimum value of $v(U)$.

Thus, U is a maximum-value feasible subset of V if and only if $U \oplus X$ is a minimum-weight good subset of V. ■

1(b)

We construct a flow network G' from G as follows: We introduce a source vertex s and add a directed edge (s, x) with capacity $c(s, x) = w(x)$ for every vertex $x \in X$. We also introduce a sink vertex t and add a directed edge (y, t) with capacity $c(y, t) = w(y)$ for every vertex $y \in Y$. For every edge joining x and y in G where $x \in X$ and $y \in Y$, we create a corresponding directed edge (x, y) in G' with capacity $c(x, y) = \infty$.

For any cut, let the set of vertices in the source be S and the set of vertices in the sink be T.

We now show that S is not a feasible subset if and only if the capacity of the cut in G' is infinite.

Proof. Note that by construction we do not have any edge of the form (y, x) in G' where $x \in X$ and $y \in Y$. If there is an edge (x, y) in G' such that $x \in X$ and $y \in Y$, then by construction the capacity of this edge is infinite. If x and y both belong to the source or both belong to the sink, then they do not add anything to the capacity of the cut. Thus the only way the capacity of the cut can be infinite is if $x \in S$ and $y \in T$. Now, $y \in \Gamma(X \cap S)$ (as $x \in X \cap S$, y is adjacent to x, and $y \notin X \cap S$) but $y \notin S$. This means that S is not a feasible subset.

Going in the other direction, if S is not a feasible subset, that means there is a vertex $u \notin X \cap S$ connected to a vertex $x \in X \cap S$, and u is not in S. The only vertices connected to x are s and vertices in Y, since G' is constructed from G which is bipartite. Since u is not in S, that means $u \neq s$. Therefore, $u \in Y$. Thus the edge connecting x and u is directed from x to u by construction of G', and $c(x, u) = \infty$. Also since $u \notin S$, therefore $u \in T$. Since $x \in S$, $u \in T$, and $c(x, u) = \infty$, therefore the cut in G' has infinite capacity. ■

This means that finding a cut with finite capacity in G' is equivalent to finding a feasible subset S which is equivalent to finding a good subset $S \oplus X$.

For convenience we let $s \in X$ and $t \in Y$. Consider a cut with finite capacity and let $S = X_1 \cup Y_1$ where $X_1 \subseteq X$ and $Y_1 \subseteq Y$. Then we denote the set of vertices in the sink by $T = X_2 \cup Y_2$ where $X_2 = X \setminus X_1$ and $Y_2 = Y \setminus Y_1$. Also $s \in S$ and $t \in T$.

We see that $S \oplus X = Y_1 \cup X_2$.

Since there are no edges between X_1, X_2, Y_1, Y_2 that contribute to the capacity of the cut, therefore the capacity of the cut comes from the edges from s and the edges to t. Edges from s go to X_1 and X_2, and edges to t come from Y_1 and Y_2. Since X_2 belongs to the sink and Y_1 belongs to the source, therefore the capacity of the cut is equal to the sum of the capacity of the edges from s to the vertices in X_2 plus the sum of the capacity of the edges from the vertices in Y_1 to t.

Or, capacity of the cut $= w(X_2) + w(Y_1) = w(S \oplus X)$.

Thus, finding the minimum cut in G' is equivalent to finding a minimum-weight good subset of V.

Thus, to compute a maximum value feasible-subset of V, we construct the flow network G' from G which takes polynomial time. We can then compute the minimum cut (S, T) of G' in polynomial time, using Edmonds-Karp and then BFS or DFS. S (minus the extra vertex s we added) is the maximum value feasible subset of V.

2

We first show that if we always discharge the highest overflowing vertex, then there are at most $|V|$ consecutive discharges which don't call the Relabel function, that is, between 2 Relabel calls, there are at most $|V|$ calls to discharge.

Proof. If no relabels occur within a DISCHARGE(u) call, then only PUSH(u, v) calls have occurred within it. Push operations do not affect the heights of any vertices, but they may change some vertices from non-overflowing to overflowing. Any vertex v which gets pushed to from u has height equal to $h(u) - 1$. So any vertex which becomes overflowing during a DISCHARGE operation between 2 relabels will not have height greater than any vertices previously discharged between those 2 relabels. Also, a vertex that has been discharged will not be discharged again unless there is a call to RELABEL. This is because for a vertex to be discharged, it has to have an overflow. Since it was discharged once, it does not have any overflow. The only way it can get an overflow is if some other vertex pushes flow to it through a PUSH operation. But this is not possible as any other overflowing vertex will have height lesser than the height of this vertex, as proved above. Hence, every vertex gets discharged at most once between 2 relabel calls. Since there are $|V|$ vertices, there are at most $|V|$ calls to discharge between 2 relabels. ■

We know that there are $O(V^2)$ relabel operations in total for any push-relabel algorithm (Corollary 26.21 in CLRS). Thus, there are at most $O(V^3)$ calls to DISCHARGE for this implementation.

We now show how to keep track of the highest-overflowing vertex using constant time operations. We keep an array A of doubly-linked lists such that $A[i]$ is a list of overflowing vertices of height i. We also have a pointer $v.position$ for every vertex v that points to where v is in the array of lists. If we want to update the height of v, then we delete v from its original position in constant time, since we can directly access its location and then delete it in constant time since it is a doubly-linked list. We then add v in constant time to the tail of the linked-list at the new height array index, and update $v.position$ to its new position in the array of lists. We keep a pointer *highest* which points to the head of the maximum height non-empty list in A. We use this pointer to access the highest overflowing vertex to discharge.

So now in the DISCHARGE function, after we call PUSH(u, v) we check if v has become overflowing, and if so we add it to A in constant time. After we call RELABEL(u) as well, we update the height of u in A. If the height of u increases, then we make *highest* point to the head of the list that that u is now in.

To analyze the work done by DISCHARGE over all its calls, we add the work done by RELABEL, PUSH, and advancing in the neighbour list of u. The work done by RELABEL is $O(VE)$ across all relabels (slide 35 in push relabel class slides), the total number of times we advance in the neighbour list of a vertex is $O(VE)$ (Theorem 26.3 in CLRS, page 759 paragrpah 4), the number of saturating pushes is $O(VE)$ (Lemma 26.22 in CLRS), and the number of non-saturating pushes is $O(V^3)$ (Lemma 26.23) . The remaining operations are constant-time in one call of DISCHARGE, so over all the calls they contribute $O(V^3)$ running time.

Adding all these together, we get a total run-time of $O(V^3)$.

4(a)

Since the edges of P are a subset of E, therefore the vertices along the path of P alternate between L and R since G is a bipartite graph. Since the first endpoint of P is in L and the last endpoint is in R, therefore P has an odd number of edges. The first edge of P starts at an unmatched vertex in L, and so the first edge belongs to $E - M$. After this, the edges alternate between M and $E - M$. So the number of edges in P is

$$\underbrace{1}_{\text{first edge from } E-M} + \underbrace{x}_{\text{edges from } M} + \underbrace{x}_{\text{edges from } E-M}$$

That is, P has $1 + x$ edges from $E - M$ and x edges from M. So the number of edges in $P - M$ is $1 + x$. If we denote the number of edges in M as $|M|$, then the number of edges in $M - P$ is $|M| - x$.

Hence, the number of edges in $M \oplus P$ = Number of edges in $(M - P) \cup (P - M) = (1 + x) + (|M| - x) = |M| + 1$.

$M \oplus P$ is a set of disjoint edges such that each edge has one endpoint in L and one in R. So no 2 edges are incident on the same vertex in G. Hence, $M \oplus P$ is a matching.

Now, since $P_1, P_2, \cdots, P_k$ are vertex disjoint, they also do not share any edges. So $(P_1 \cup P_2 \cup \cdots \cup P_k)$ is a set of disjoint augmenting paths in G.

$M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)$ is a set of disjoint edges such that each edge has one endpoint in L and one in R. So no 2 edges are incident on the same vertex in G. Hence, it is a matching.

The number of edges in

$$P_1 = \underbrace{x_1}_{\text{edges from } M} + \underbrace{1 + x_1}_{\text{edges from } E-M}$$

$$P_2 = \underbrace{x_2}_{\text{edges from } M} + \underbrace{1 + x_2}_{\text{edges from } E-M}$$

$$\vdots$$

$$P_k = \underbrace{x_k}_{\text{edges from } M} + \underbrace{1 + x_k}_{\text{edges from } E-M}$$

Thus the number of edges in $(P_1 \cup P_2 \cup \cdots \cup P_k)$ is

$$\underbrace{x_1 + x_2 + \cdots + x_k}_{\text{edges from } M} + \underbrace{k + x_1 + x_2 + \cdots + x_k}_{\text{edges from } E-M}$$

The number of edges in $M - (P_1 \cup P_2 \cup \cdots \cup P_k)$ is

$$|M| - (x_1 + x_2 + \cdots + x_k) \tag{1}$$

The number of edges in $(P_1 \cup P_2 \cup \cdots \cup P_k) - M$ is

$$k + x_1 + x_2 + \cdots + x_k \tag{2}$$

$\therefore$ the number of edges in $M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k) = (1) + (2) = |M| + k$.

4(b)

The degree of every vertex in M is at most 1. The degree of every vertex in M^* is also at most 1. In $M \oplus M^*$, every vertex that is matched by both M and M^* (with a different edge from M and M^*) will have degree 2. Vertices that are matched by exactly 1 of M and M^* will have degree 1, and vertices that are matched by neither M not M^* or matched by both will have degree 0 in $M \oplus M^*$. Hence, every vertex in the graph $G' = (V, M \oplus M^*)$ has degree at most 2.

In simple paths and cycles, every vertex has degree at most 2. Thus, G' is a disjoint union of simple paths and cycles.

Let us suppose that 2 consecutive edges belong to the same matching, say M. This means that the common vertex of these 2 edges has degree 2 in the matching M. But this goes against the definition of a matching, and hence M cannot be a matching which is a contradiction. Thus, consecutive edges in a simple path or cycle have to come from different matchings, i.e. the edges in a simple path or cycle belong alternately to M or M^*.

In cycles of $M \oplus M^*$ the number of edges from M = number of edges from M^*. So if $|M| < |M^*|$, then in $M \oplus M^*$ $|M^*| - |M|$ edges from M^* show up in the disjoint simple paths. Each simple path can have at most one more edge from one matching than from another. So there will be at least $|M^*| - |M|$ disjoint simple paths which have one more edge from M^* than from M. These paths will all be disjoint, and will have one end point in L and one in R as they have an odd number of edges and alternate between edges in M and edges not in M. Their endpoints are matched by M^* and not M. Hence each of these paths are augmenting with respect to M, and there are $|M^*| - |M|$ of them.

4(c)

We first show that if a vertex is unmatched by M', then it is also unmatched by M.

Proof. Consider a vertex that is matched by an edge in M. There are two cases for the augmenting paths:

Case 1: If none of the augmenting paths from P_1 to P_k contain that edge, then this edge belongs to $M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)$. Thus, the vertex is matched by M'.

Case 2: If one of the augmenting paths from P_1 to P_k contain this edge (More than one path cannot contain this edge, as the paths are vertex-disjoint) then this is a matched edge within the augmenting path. Matched edges always occur in the middle of augmenting paths, that is, an augmenting path will never begin or end with a matched edge (as the augmenting path starts and ends with an unmatched vertex). Thus, there will be another edge belonging to the augmenting path incident on this vertex. This second edge will not belong to M, as the edges in augmenting paths alternate between M and $E - M$. Thus, this edge belongs only

to the augmenting path and not to M and so it belongs to $M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)$. Thus, this edge belongs to M', and the vertex under consideration is matched by M'.

Thus, if a vertex is matched by M, then it is also matched by M'. This means that if a vertex is unmatched by M', then it is also unmatched by M. ■

Now, M' contains edges that either belong only to M, or belong only to $P_1 \cup P_2 \cup \cdots \cup P_k$. P is vertex disjoint from $P_1, P_2, \cdots, P_k$ so P does not contain any edge from $P_1, P_2, \cdots, P_k$. Thus, any edge of P in M' must be an edge in M. Also, any edge of P not in M' is an edge not in M. The edges of P alternate between M' and $E - M'$, which is equivalent to saying they alternate between M and $E - M$. P starts at a vertex in L unmatched by M', and ends at a vertex in R unmatched by M'. This means that P starts at a vertex in L unmatched by M, and ends at a vertex in R unmatched by M. Thus, P is an augmenting path with respect to M. This means that the length of P has to be greater than l, as $P_1, P_2, \cdots, P_k$ is a maximal set of vertex-disjoint augmenting paths of length l with respect to M. Thus, P has more than l edges.

4(d)

$$\begin{aligned} A &= (M \oplus M') \oplus P \\ &= (M \oplus (M \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)) \oplus P \\ &= ((M \oplus M) \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)) \oplus P \\ &= (\phi \oplus (P_1 \cup P_2 \cup \cdots \cup P_k)) \oplus P \\ &= (P_1 \cup P_2 \cup \cdots \cup P_k) \oplus P \end{aligned}$$

From the results of part a, we get that $|M'| = |M| + k$, and since M' is a matching, $|M' \oplus P| = |M'| + 1 = |M| + k + 1$. Now, $(M \oplus M') \oplus P = M \oplus (M' \oplus P)$. From part a, $M' \oplus P$ is a matching. Since M and $M' \oplus P$ are matchings, and $|M| < |M| + k + 1 = |M'|$, we can apply the result of part b, and we get that the set of edges $(M \oplus M') \oplus P$ contains at least $|M'| - |M| = k + 1$ vertex-disjoint augmenting paths with respect to M. We know that the length of a shortest-disjoint augmenting path with respect to M is l, and hence the number of edges in $A \geq (k+1)\, l$.

$A = (P_1 \cup P_2 \cup \cdots \cup P_k) \oplus P$ and each of $P_1, P_2, \cdots, P_k$ have l edges. Also, as P is not vertex disjoint from $P_1, P_2, \cdots, P_k$ it shares at least one edge with $(P_1 \cup P_2 \cup \cdots \cup P_k)$. Thus

$$kl + (\text{number of edges in } P) - 2 \geq kl + l$$

$$(\text{number of edges in } P) \geq l + 2$$

So the number of edges in P is more than l.

4(e)

We denote the maximum matching by M^*. Let us suppose that if possible, $|M^*| > |M| + |V|/(l+1)$. Then $|M^*|-|M| > |V|/(l+1)$, or $|M^*|-|M| \geq |V|/(l+1)+1$. Since $|M^*| > |M|$, from the result of part b we get that $M \oplus M^*$ contains at least $|M^*| - |M|$ vertex-disjoint augmenting paths with respect to M. So $M \oplus M^*$ contains at least $|V|/(l+1) + 1$ vertex-disjoint augmenting paths with respect to M. Or, $M \oplus M^*$ contains more than $|V|/(l+1)$ vertex-disjoint augmenting paths with respect to M. Since the shortest augmenting path with respect to M has l edges, it will be incident on $l+1$ vertices. Since there are more than $|V|/(l+1)$ **vertex-disjoint** augmenting paths with respect to M and each of these is incident on at least $l+1$ vertices, we get the total number of unique vertices on which these paths are incident is more than $(|V|/(l+1)) \cdot (l+1) = |V|$. But we only have $|V|$ vertices in the graph, and we are getting that there are more than $|V|$ vertices on which the augmenting paths are incident. This is a contradiction, and hence the size of the maximum matching M^* is at most $|M| + |V|/(l+1)$.

4(f)

Suppose we are at iteration $\sqrt{|V|}$ and we have matching M. The length of the shortest augmenting path with respect to M is at least $\sqrt{|V|}$ because in every iteration we find the maximal set of vertex-disjoint shortest augmenting paths with respect to the matching at that iteration, and the shortest augmenting path length increases by at least 1 each iteration.

Since the length of the shortest augmenting path with respect to M is at least $\sqrt{|V|}$, any augmenting path with respect to M will be incident on at least $\sqrt{|V|} + 1$ vertices. Since there are $|V|$ vertices in the graph, the number of vertex-disjoint augmenting paths with respect to M is less than $\sqrt{|V|}$.

Let the maximum matching for the graph be M^*. Then $|M^*| \geq |M|$. So from part b, we get that $M^* \oplus M$ contains at least $|M^*| - |M|$ vertex-disjoint augmenting paths with respect to M. From above, we get that $|M^*| - |M| < \sqrt{|V|}$. Since the cardinality of the matching increases by at least 1 every iteration, there are at most $|M^*| - |M|$ iterations left to reach the maximum matching. Thus, the number of iterations left to reach M^* is less than $\sqrt{|V|}$. So total number of repeat loop iterations is at most $2\sqrt{|V|}$.

4(g)

We first want to find the length l of the shortest augmenting path, so for that we add a source vertex s and connect it to each of the unmatched vertices in L. Then we run a BFS from this source, with the constraint that starting from the second edge, edges have to alternate between $E - M$ and M. If we hit an unmatched vertex in R for the first time, and the length

of the path from s to this vertex is k, then we stop BFS when we have reached level $k-1$ (assuming levels are indexed from 0). Essentially, we have run BFS in parallel from each of the unmatched vertices in L to find l.

Now for each of the unmatched vertices that we reached in R, we trace back along the path of unmarked edges and vertices taken to reach that vertex by BFS, marking the edges as belonging to the set $\mathcal{P}$ of vertex disjoint augmenting paths, and marking the vertices as well. In this way we get our set $\mathcal{P}$, and due to our system of marking vertices so we don't traverse them again, we are assured that all the paths in $\mathcal{P}$ are vertex disjoint. This process takes $O(E)$ time.

The total running time of HOPCROFT-KARP is the time to run a single iteration of the repeat loop multiplied by the number of repeat loop iterations. The symmetric set difference operation can be carried out in $O(E)$ time, and the time to find the maximal set of vertex-disjoint shortest augmenting paths with respect to M is $O(E)$. Thus the total run time of the algorithm is $2\sqrt{|V|} \cdot O(E) = O(\sqrt{V}E)$.